

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Брянский государственный технический университет

Факультет информационных технологий

Кафедра «Компьютерные технологии и системы»

Дисциплина «Объектно-ориентированное программирование»

**КУРСОВОЙ ПРОЕКТ**  
на тему: разработка игры «Арканоид»

|  |  |
| --- | --- |
|  | Выполнил:  студент группы О-19-ИАС-аид-С  Колотвин А.П.  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |
|  | Проверил:  к.т.н., доц. Чмыхов Д.В.  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |

Брянск 2021

## СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc75113167)

[1. АНАЛИТИЧЕСКИЙ РАЗДЕЛ 7](#_Toc75113168)

[1.1. Описание предметной области 7](#_Toc75113169)

[1.2. Обзор и анализ существующих программных решений 9](#_Toc75113170)

[1.2.1. Игра «Арканоид» от Play Machine 9](#_Toc75113171)

[1.2.2. Игра «Арканоид Онлайн» 10](#_Toc75113172)

[1.3. Функциональные требования к разрабатываемому программному продукту 11](#_Toc75113173)

[2. КОНСТРУКТОРСКИЙ РАЗДЕЛ 13](#_Toc75113174)

[2.1. Обоснование выбранной среды и языка программирования 13](#_Toc75113175)

[2.2. Функциональная схема работы программы (IDEF0) 15](#_Toc75113176)

[2.3. Диаграмма потоков данных (DFD) 17](#_Toc75113177)

[2.4. Диаграмма объектов и диаграмма классов 18](#_Toc75113178)

[2.5. Описание используемых методов и алгоритмов 21](#_Toc75113179)

[2.6. Выбор графического и пользовательского интерфейса 22](#_Toc75113180)

[2.6.1. Первый вариант интерфейса программы 23](#_Toc75113181)

[2.6.2. Второй вариант интерфейса программы 24](#_Toc75113182)

[3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 27](#_Toc75113183)

[3.1. Описание структуры и состав программной среды 27](#_Toc75113184)

[3.2. Разработка алгоритмов отдельных подзадач 30](#_Toc75113185)

[3.3. Руководство пользователя 32](#_Toc75113186)

[3.4. Подготовка программы к работе и условия её эксплуатации 33](#_Toc75113187)

[4. ЭКСПЕРИМЕНТАЛЬНЫЙ РАЗДЕЛ 35](#_Toc75113188)

[4.1. Виды контроля качества разрабатываемого ПО 35](#_Toc75113189)

[4.2. Методика проведения и результаты тестирования 36](#_Toc75113190)

[4.3. Методы и способы устранения ошибок 37](#_Toc75113191)

[4.4. Отладка выявленных ошибок, обнаруженных при тестировании 38](#_Toc75113192)

[ЗАКЛЮЧЕНИЕ 40](#_Toc75113193)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 41](#_Toc75113194)

[ПРИЛОЖЕНИЯ 43](#_Toc75113195)

[Приложение 1. Листинг программы 43](#_Toc75113196)

[Приложение 2. Графический интерфейс программы 56](#_Toc75113197)

## ВВЕДЕНИЕ

Во все времена человек стремился стать успешнее в выбранной им профессиональной сфере, чтобы обеспечить себе счастливую и беззаботную жизнь. Он учится на протяжении всей своей жизни, приобретает всякий раз новые знания и совершенствуется. Но на пути к своей мечте, как всегда и бывает, человек сталкивается с различными трудностями. Конечно же, чтобы достичь желаемого, эти трудности необходимо преодолевать. Для этого затрачивается масса времени и сил, которые имеют свойства истощаться. В момент такого истощения необходимо уметь переключать своё внимание на что-то другое, например, занятие спортом или развлечения. Если человек не будет отдыхать, рано или поздно произойдет его эмоциональное перегорание, что повлечет за собой неприятные последствия и достижение его цели будет уже невозможна.

Конечно, эта проблема будет актуальна всегда, поэтому человечество всякий раз придумывает новые виды проведения досуга. Одним из таких видов являются компьютерные видеоигры. Компьютерные игры давно вошли в жизнь современного человека и заняли практически первое место среди всех видов отдыха. И это можно объяснить: виртуальная реальность имеет свойство притягивать своими неограниченными возможностями, а разработчики не перестают радовать любителей игр новыми продуктами.

Многие считают, что современные компьютерные игры не представляют собой абсолютно никакой пользы, оказывая негативное влияние, как на наше сознание, так и на физическое состояние. Однако учёными было доказано позитивное влияние видеоигр на организм человека. Благодаря видеоиграм улучшаются такие человеческие характеристики, как память, воображение, внимание, логика и быстрота реакции. Видеоигры способны развивать мышечную память, и самое главное, снимают стресс, что благоприятно сказывается на психическом здоровье.

Задачами являются: приобретение практических навыков в работе с основными принципами объектно-ориентированного программирования, изучение новых средств языка программирования C#, анализ предметной области, приобретение навыков в построении схем, диаграмм и пользовательского интерфейса.

Таким образом, целью данного курсового проекта является разработка такого программного комплекса, который помог бы человеку приятно провести время с пользой для своего здоровья, снять напряжение и развить его внимание, моторику и быстроту реакции.

## Техническое задание

на курсовой проект по дисциплине «Объектно-ориентированное программирование».

**Студент** Колотвин А.П. **Группа** О-19-ИАС-аид-С

### Наименование и область применения

Полное наименование системы: «Игра Арканоид».

### Основания для разработки

Основанием для разработки проекта является демонстрация теоретических и практических навыков, приобретенных за курс обучения, по дисциплине «Объектно-ориентированное программирование».

### Назначение разработки

Назначением разработки является создание игры «Арканоид» на языке программирования C#, предназначенной для развития реакции и моторики пользователя, а также для приятного времяпрепровождения.

### Технические требования к программе или программному изделию

Программа должна включать в себя следующие функциональные характеристики:

* должно быть реализовано управление пользователем игровой платформой;
* должна быть реализована физика полёта и столкновения шарика о другие объекты (границы игрового поля, игровую платформу, блоки);
* необходимо реализовать счётчик «жизней» у игрока за один сеанс;
* некоторые блоки при столкновении с шариком должны иметь уровень прочности (ломаются при более одном столкновении);
* должны быть реализованы случайные «бонусы» (дополнительная «жизнь», увеличение платформы и др.);
* должен быть реализован конструктор пользовательских игровых уровней;
* необходимо реализовать счётчик заработанных очков.

### Требования к надёжности

Надежное функционирование программы достигается при использовании лицензионного программного обеспечения, удовлетворяющего системным требованиям, организации бесперебойного питания технических средств. Ошибки, возникающие при некорректных действиях в работе с программой пользователем недопустимы.

### Требования к составу и параметрам технических средств

* процессор Intel Core i3 или его аналоги;
* не менее 4 Гб оперативной памяти;
* объем видеопамяти от 2 Гб и выше;
* 64 – разрядная операционная система Windows 10 версии.
* устройства, необходимые для взаимодействия пользователя с программой, такие как клавиатура и мышь.

### Требования к информационной и программной совместимости

Программа должна быть написана на языке C# в приложении Windows Forms. Она разрабатывается под платформу Windows и должна работать под её основными версиями.

**Руководитель Чмыхов Д.В.**

## АНАЛИТИЧЕСКИЙ РАЗДЕЛ

### Описание предметной области

В настоящее время компьютерные игры занимают лидирующее место среди остальных видов отдыха. Это объясняется их огромным разнообразием жанров, уникальных историй и сюжетов, что дает возможность каждому человеку выбрать то, что ему по душе. История компьютерных игр начинается в 1940-х и 1950-х годах, когда в академической среде разрабатывались простые игры и симуляции. Компьютерные игры длительное время не были популярны, и только в 1970-х и 1980-х годах, когда появились для широкой публики аркадные автоматы, игровые консоли и домашние компьютеры, компьютерные игры становятся частью поп-культуры.

За долгие годы развития видеоигр их разновидность стала настолько велика, что едва получится дать полную классификацию. Однако основными жанрами компьютерных игр были выделены:

* аркады;
* головоломки;
* гонки;
* квесты;
* стратегии;
* шутеры;
* симуляторы.

Аркады имеют несложный игровой процесс, не меняющийся с течением времени. В большинстве аркадных игр для достижения результатов игроку нужно проявлять хорошую реакцию. Обычно в таких играх развита система бонусов: начисление очков, временное улучшение характеристик персонажа и т.д.

Головоломки – это такой вид компьютерных игр, для прохождения которых от игрока требуется решить задачи, требующие логики, воображения и интуиции.

Гонки – один из наиболее популярных видов игр, в которых игроку, движущемуся на транспортном средстве, нужно первым прийти к финишу. Наиболее популярным транспортным средством в гонках является автомобиль, однако существует множество гонок и на других средствах передвижения, вплоть до космических кораблей.

Квесты, или игры-приключения – обычно игры, в которых герой продвигается по сюжету, выполняет различные задания и взаимодействует с игровым миром посредством использования предметов, общения с другими персонажами (NPC) и решения головоломок.

Симуляторы – вид компьютерных игр, имитирующих управление каким-либо процессом, аппаратом или средством передвижения. Целью каждого симулятора является максимально приблизить игрока к условиям управления реальным объектом.

Стратегии требуют от игрока планирования и выработки определенной тактики и стратегии для достижения необходимой цели, например, победы в военной операции или захвата вражеского государства. Игрок управляет не единственным персонажем, а армиями, городами, государствами или даже цивилизациям.

Шутеры – это вид игр, в котором игроку, в большинстве случаев действуя в одиночку, нужно уничтожать противников при помощи различного оружия.

Одним из ярких представителей жанра аркадных игр является «Арканоид». Это видеоигра для игровых автоматов, разработанная компанией Taito в 1986 году. Игра основана на играх серии Breakout («прорыв»). Именно её название стало нарицательным для класса подобных игр.

Суть игры заключается в том, что игрок контролирует небольшую платформу-ракетку, которую можно передвигать горизонтально от одной стенки до другой, подставляя её под шарик, предотвращая его падение вниз. Удар шарика по кирпичу приводит к разрушению кирпича. После того как все кирпичи на данном уровне уничтожены, происходит переход на следующий уровень, с новым набором кирпичей. Есть и некоторое разнообразие: определённые кирпичи нужно ударять несколько раз, иногда появляются летающие враги, от которых отталкивается шарик, удар по некоторым кирпичам приводит к выпадению из них капсул-призов — приз активируется, если поймать такую капсулу ракеткой.

### Обзор и анализ существующих программных решений

#### Игра «Арканоид» от Play Machine

. Игра «Арканоид» от сервиса Play Machine выглядит следующим образом (рис.1.1):

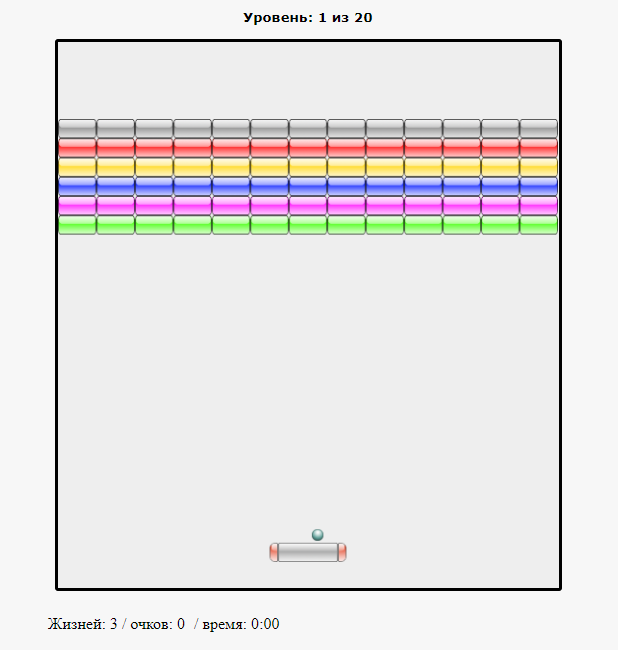


Рис.1.1. Игра «Арканоид» от Play Machine

Для определения основных функциональных возможностей требуемых для реализации и выявлений необходимых конкурентных преимуществ разрабатываемого программного комплекса, выделим основные достоинства и недостатки у данного программного решения.

Достоинства:

* разнообразие игровых бонусов;
* реализовано несколько игровых уровней;
* наличие правил игры «Арканоид».

Недостатки:

* палитра цветов тусклая, при длительном сеансе глаз устает и замыливается;
* неудобное расположение счетчика очков и игровых жизней;
* прочные блоки (которые ломаются от более одного столкновения) плохо реализованы;
* управление ракеткой резкое и неконтролируемое.

#### Игра «Арканоид Онлайн»

«Arcanoid Online» - это бесплатная браузерная игра, не требующая установки и регистрации, вдохновлённая классическими играми Brick Breaker, которая работает на настольных компьютерах и смартфонах (рис.1.2).

Достоинства:

* счетчик игровых очков расположен на видном месте;
* разнообразие игровых бонусов;
* яркая палитра цветов.

Недостатки:

* отсутствует счетчик игровых жизней;
* отсутствует руководство для пользователя с правилами игры;
* много свободного пространства, из-за чего игра кажется затянутой и скучной;
* прочные блоки (которые ломаются от более одного столкновения) не реализованы;
* управление ракеткой резкое и не отзывчивое.

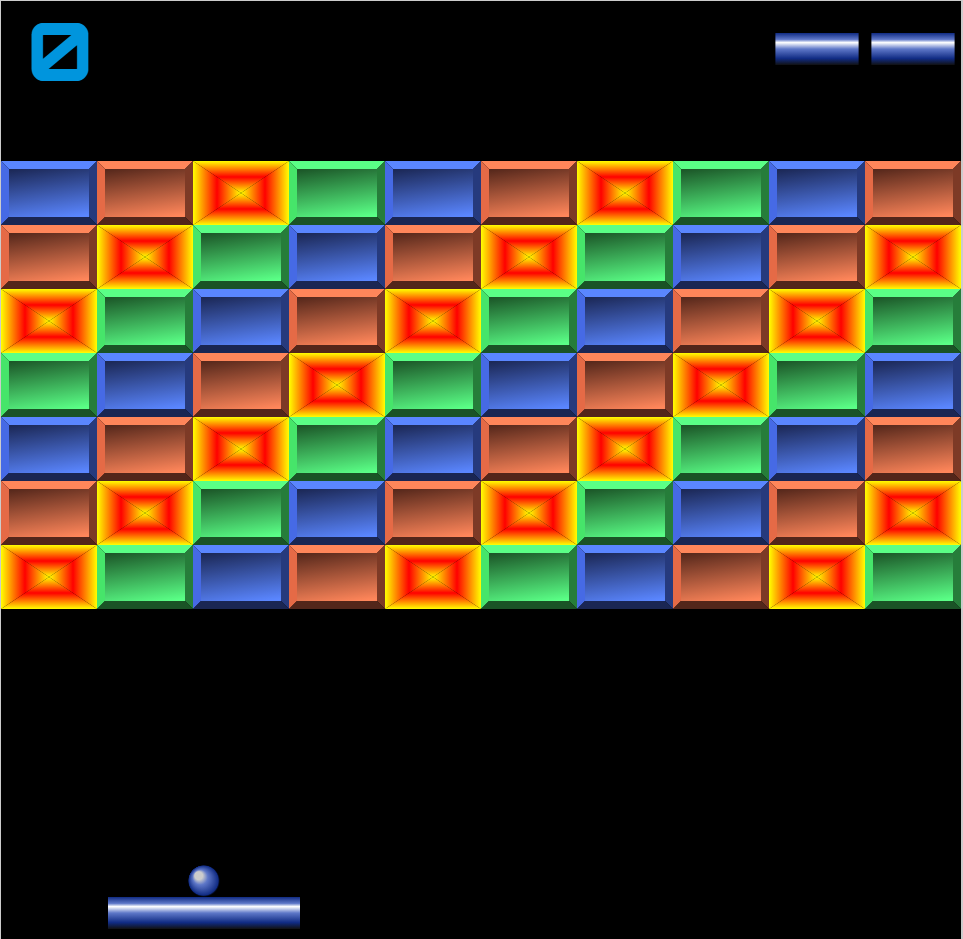


Рис.1.2. Игра «Arcanoid Online»

Таким образом, на основе анализа данных продуктов, разрабатываемый программный комплекс будет обладать всеми перечисленными достоинствами, а все выявленные недостатки будут устранены.

### Функциональные требования к разрабатываемому программному продукту

На основе проделанного анализа предметной области и существующих программных решений, необходимо выделить набор функциональных требований к разрабатываемому программному продукту.

Таким образом, программный комплекс должен включать в себя следующий набор функциональных требования:

* палитра цветов должна быть яркой и приятной;
* должны быть реализованы счетчики игровых жизней и заработанных очков;
* должны быть реализованы несколько игровых уровней и бонусов;
* управление ракеткой должно быть плавным и отзывчивым;
* должно быть реализовано разрушение прочных блоков (ломаются от более одного попадания);
* свободное игровое пространство должно быть минимально, что игровой процесс не казался затянутым;
* программа должна содержать руководство пользователя.

## КОНСТРУКТОРСКИЙ РАЗДЕЛ

### Обоснование выбранной среды и языка программирования

Для выполнения курсового проекта и создания программного продукта было решено использовать язык программирования C# и среду программирования Microsoft Visual Studio.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемых кодах для всех платформ, поддерживаемых Windows, Windows Mobile, .NET Framework. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных.

У этой среды разработки множество достоинств. Например, поддержка множества языков при разработке. Далее к достоинствам можно отнести интуитивный стиль кодирования. По умолчанию Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев. Такие незначительные отличия делают код более удобным для чтения и менее подверженным ошибкам. Предлагаемые в Visual Studio инструменты отладки являются наилучшим средством для отслеживания загадочных ошибок и диагностирования странного поведения. Разработчик может выполнять свой код по строке за раз, устанавливать интеллектуальные точки прерывания, при желании сохраняя их для использования в будущем, и в любое время просматривать текущую информацию из памяти. Данная среда программирования, безусловно, обладает массой плюсов, поэтому можем сказать, что её выбор является полностью оправданным.

Далее рассмотрим непосредственно сам язык, на котором будет создаваться наш проект. Язык программирования C# – объектно-ориентированный язык программирования. Разработан в 1998–2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Переняв многое от своих предшественников – языков C++, Pascal, Модула, Smalltalk и, в особенности, Java – С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ и некоторых других языков, не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов). Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку (в том числе операторов явного и неявного приведения типа), атрибуты, события, свойства, обобщённые типы и методы, исключения, комментарии в формате XML.

Рассмотрим несколько важных достоинств этого языка:

* данный язык хорошо развивается благодаря постоянной поддержки компании Microsoft;
* в данном языке много синтаксического сахара – конструкций, которые созданы для облегчения написания и понимания программного кода, что особенно важно для облегчения чтения кода другими программистами;
* синтаксис языка похож на C, C++ или Java, поэтому это значительно облегчает переход других программистов. В то же время это один из самых перспективных языков для начинающих программистов.

### Функциональная схема работы программы (IDEF0)

IDEF0 – это методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность.

Данный стандарт позволяет представить программный комплекс в виде набора функциональных блоков, каждый из которых может осуществлять взаимодействие с другими функциональными блоками посредством четырех видов интерфейсов (входа, управления, механизма и выхода).

Стандарт IDEF0 предполагает использование принципа декомпозиции, который применяется при разбиении сложной задачи на составляющие её функции. Декомпозиция позволяет представить модель системы в виде иерархической структуры отдельных диаграмм. Диаграмма первого уровня декомпозиции игры «Арканоид» имеет следующий вид (рис.2.1):

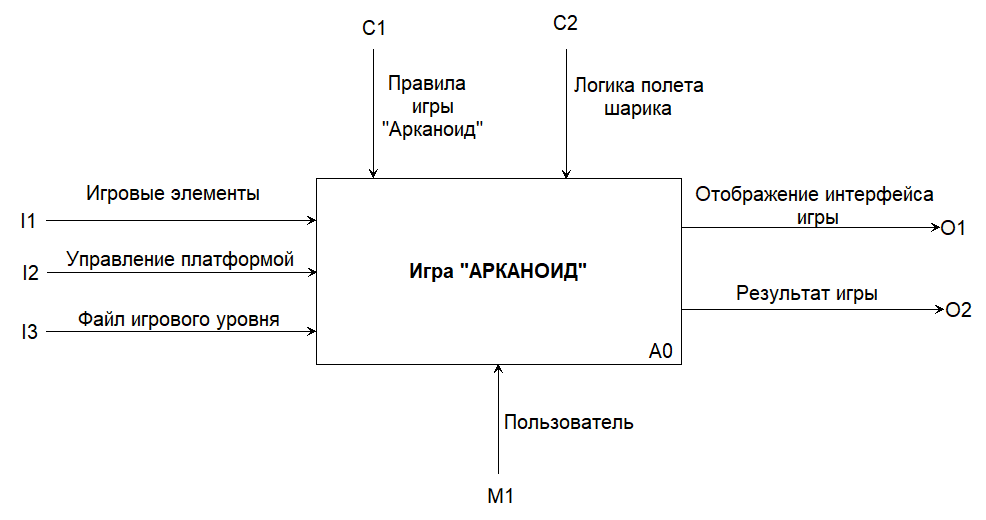


Рис.2.1. Диаграмма IDEF0 первого уровня декомпозиции

Интерфейс входа (input) описывает исходные данные или объекты для выполнения функции, результат выполнения которой описан в интерфейсе выхода (output). На данной диаграмме на вход поступают игровые элементы, управление игровой платформой и файл игрового уровня, содержащего блоки. На выходе – отображение интерфейса и результат игры (победа, поражение). Сверху описывается интерфейс управления (control), который описывает правила и ограничения, которыми программа руководствуется. На диаграмме это стандартные правила игры «Арканоид» и логика полёта шарика. Последний интерфейс, который описывается снизу – это интерфейс механизма (mechanism). Он описывает ресурсы, используемые в процессе выполнения функции, при этом эти ресурсы не должны изменяться. На диаграмме этим механизмом является пользователь.

На диаграмме второго уровня декомпозиции (рис.2.2) эта система разбивается на отдельные компоненты.

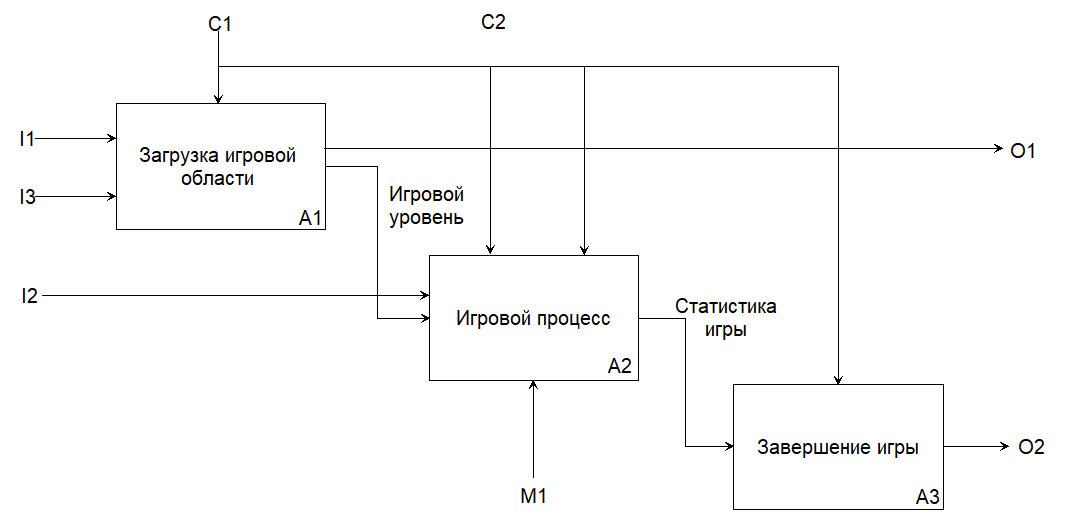


Рис.2.2. Диаграмма IDEF0 второго уровня декомпозиции

Для блока А1 входными данными являются игровые элементы и файл игрового уровня. Управление этим блоком регулируется правилами игры «Арканоид», выходными данными блока А1 является отображение игрового интерфейса и настройки, которые поступают в блок А2.

Для блока А2 входными данными являются настройки, полученные из блока А1 и управление игровой платформой. Механизмом для данного блока является пользователь. Интерфейс управления включается в себя правила игры «Арканоид» и логику полёта шарика. Выходными данными здесь являются данные об игровом процессе, которые поступают на вход в блок А3, в управление которого также входят правила игры «Арканоид» и на выходе пользователь получается результат игры.

### Диаграмма потоков данных (DFD)

Диаграмма потоков данных – это один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML. При создании диаграммы потоков данных применяется стандарт DFD (data flow diagrams). Стандарт предполагает использование четырёх элементов: функций, потоков данных, хранилищ данных и внешних сущностей.

DFD – общепринятое сокращение диаграммы потоков данных, так называется методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Модель DFD, как и большинство других структурных моделей является иерархичной, то есть, каждый процесс может быть подвергнут декомпозиции. Первый уровень декомпозиции диаграммы потоков данных для игры «Арканоид» приведен на рис.2.3.

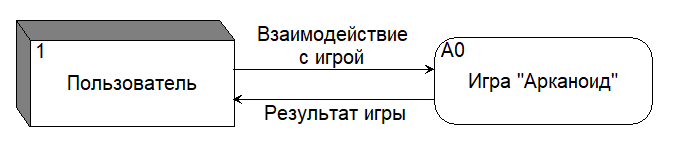


Рис.2.3. Диаграмма потоков данных первый уровень декомпозиции

Внешняя сущность – это внешний по отношению к системе объект, обменивающийся с нею потоками, в данной диаграмме этой сущностью является пользователь. Функции представляют собой действия, выполняемые на данном этапе рабочего процесса, они также преобразуют входные потоки данных в выходные в соответствии с алгоритмом работы программы.

Блок А0 на входе получает от пользователя различные команды, а на выходе передает пользователю результат игрового процесса.

В процессе декомпозиции каждая функция подвергается детализации на другой диаграмме, которая также может быть детализирована путём аналогичной декомпозиции её функций.

На втором уровне декомпозиции (рис.2.4) блок А0 подвергается детализации.

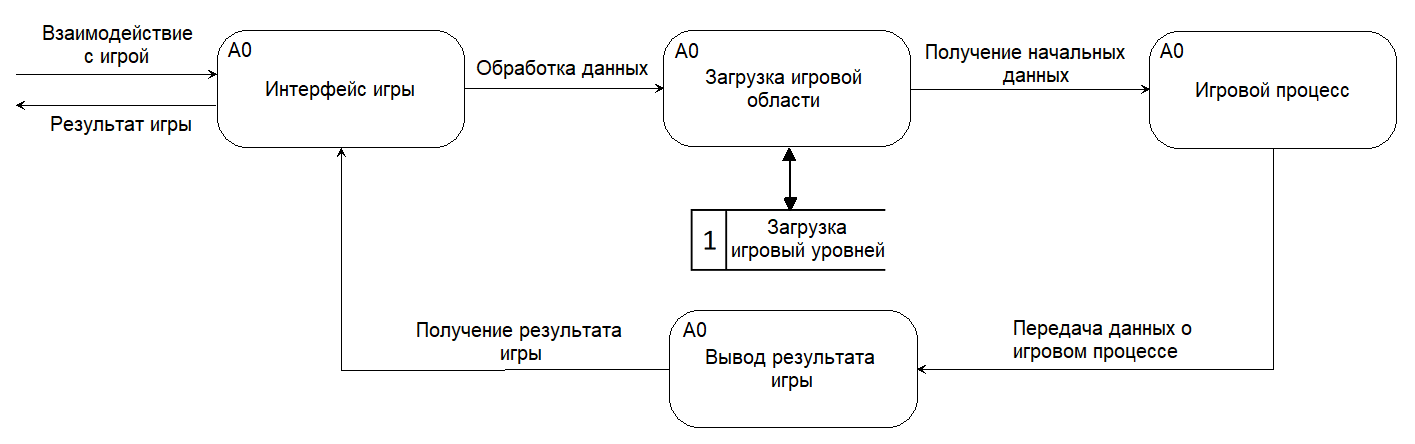


Рис.2.4. Диаграмма потоков данных второй уровень декомпозиции

Диаграммы потоков данных (DFD) обеспечивают удобный способ описания передаваемой информации, как между частями моделируемой системы, так и между системой и внешним миром.

### Диаграмма объектов и диаграмма классов

При разработке программного обеспечения с использованием принципов объектно-ориентированного программирования выполняется объектная декомпозиция, то есть, выделение в системе отдельных сущностей, которые можно представить в виде объектов. Объекты, или сущности – это абстракции, которые являются основными элементами модели, которые на диаграмме соединяются связями. Диаграмма объектов, в отличие от диаграммы классов, не представляет собой абстрактную модель, а близка к реальному поведению системы.

Диаграмма объектов для игры «Арканоид» (рис.2.5) показывает сущности и их логические связи в системе.

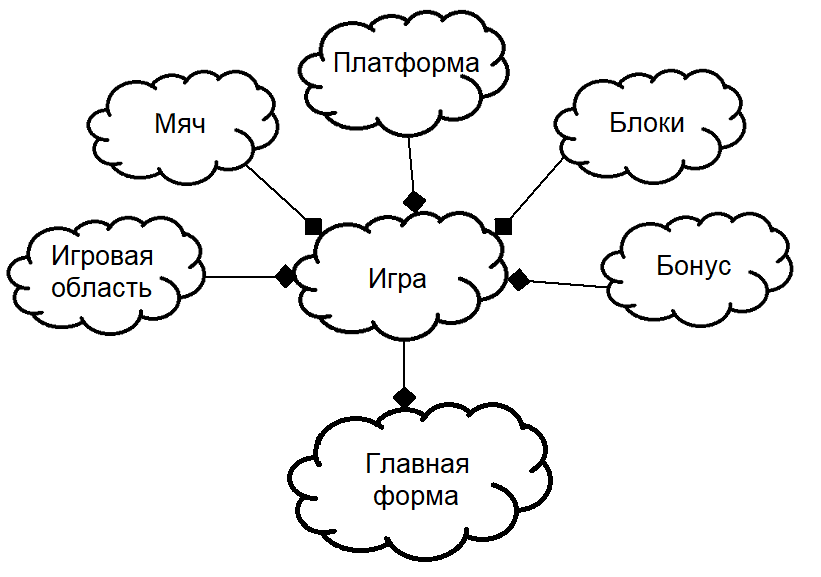


Рис.2.5. Диаграмма объектов

Сущность «Игровая область» представляет границы зоны, в которой происходит весь игровой процесс.

Сущность «Мяч» представляет игровой шар, который перемещается внутри игровой области и изменяет своё направление при контакте с другими игровыми элементами.

Сущность «Платформа» представляет ракетку, которой управляет игрок.

Сущность «Блоки» представляет собой игровые блоки, которые создаются в игровой области и разрушаются при столкновении с мячом.

Сущность «Бонус» отвечает за игровые бонусы, которые случайно генерируются при разрушении блоков, дающие игроку преимущества в ходе игры.

Сущность «Игра» является главным логическим звеном всей системы, которая связывает все игровые элементы и отвечает за корректную работу игрового процесса.

Сущность «Главная форма» хранит данные обо всем проекте и отвечает за интерфейсную (внешнюю) составляющую программы.

Диаграмма классов (Static Structure diagram) – диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Диаграмма классов (рис.2.6) является ключевым элементом в объектно-ориентированном моделировании.

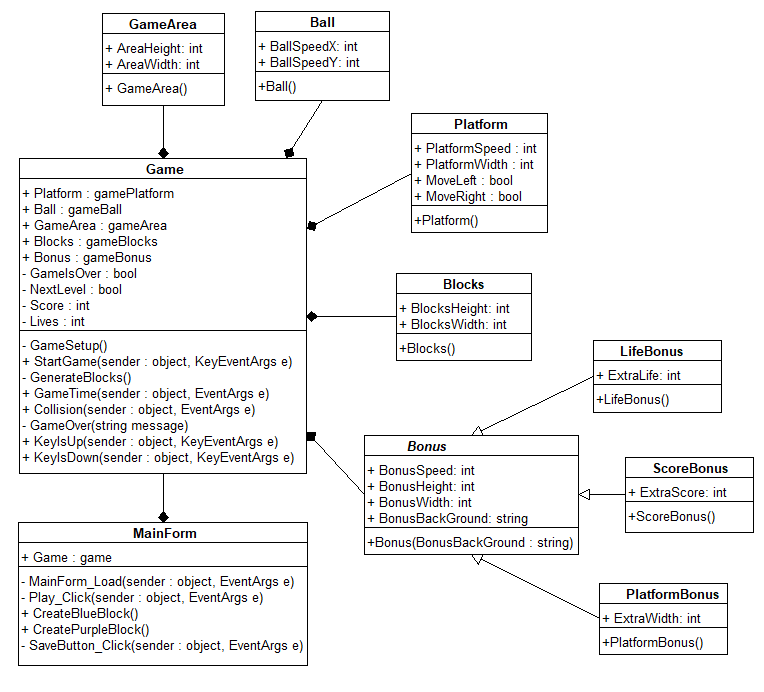


Рис.2.6. Диаграмма классов

Класс Game содержит ссылки на все игровые объекты и методы, реализующие весь игровой процесс.

Класс GameArea хранит свойства игровой области – её длину и высоту. Класс Ball содержит скорость полёта игрового мяча по оси X и Y. Класс Platform содержит свойства игровой платформы, которой управляет игрок, а именно её скорость, ширину и перемещение по игровой области. Класс Blocks хранит общие свойства создаваемых игровых блоков, а именно их ширину и высоту. Класс Bonus является абстрактным, хранит скорость полёта бонусов и их размеры. От него наследуются классы, представляющие конкретные блоки: дополнительные очки, дополнительная жизнь и увеличение платформы.

### Описание используемых методов и алгоритмов

На этапе проектирования проводится анализ работы программного комплекса в целом. Для решения отдельных задач, возникающих при создании ПО, необходимо выбрать существующие или создать свои методики и алгоритмы, которые требуется представить в формализованном виде. Например, для представления алгоритмов можно использовать блок-схемы (рис.2.7).

При запуске программы пользователь либо начинает игровой сеанс, либо создает пользовательские уровень в конструкторе, добавляя разные блоки и затем сохраняя расположение блоков в документе. В начале игрового сеанса устанавливаются начальные игровые настройки (начальное расположение игровых элементов), затем происходит генерация начального уровня. Во время игры отслеживается, прошел ли игрок все уровни, если да, то на экран выводится результат игры, иначе отслеживается положение мяча. Если мяч находится за пределами игровой области, то есть, мяч упал, то игрок теряет жизнь, в случае если игрок потерял все жизни, он терпит поражение, иначе игра продолжается. Также отслеживается, были ли разрушены все игровые блоки, в этом случае игрок переходит на новый уровень, который загружается из внешнего файла. Также при разрушении блока может генерироваться случайный игровой бонус.

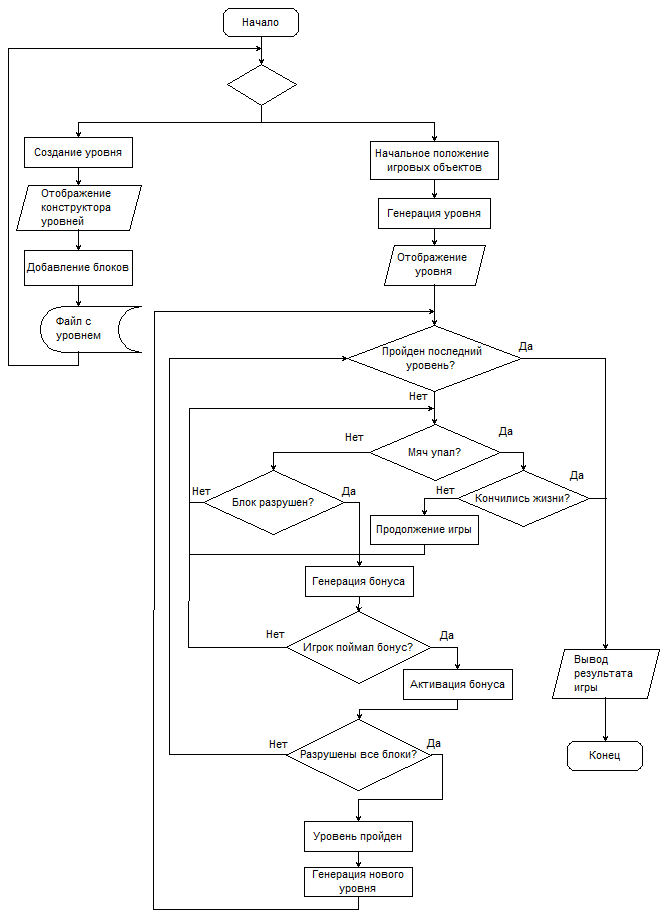


Рис.2.7. Блок-схема игры «Арканоид»

### Выбор графического и пользовательского интерфейса

Разработка графического и пользовательского интерфейса является не менее важной частью создания программного продукта. Интерфейс пользователя (user interface или сокращенно UI) – это интерфейс, с помощью которого человек может управлять программным обеспечением или аппаратным оснащением. UI должны быть удобными в использовании, чтобы взаимодействие с ними происходило на максимально интуитивном уровне. Он представляет собой окно, в котором содержатся различные элементы управления, взаимодействие с которыми осуществляется с помощью мыши или клавиатуры. Это окно представляет собой нечто вроде шлюза между пользователем и программным обеспечением. В ходе разработки программного комплекса было создано несколько вариантов графического интерфейса.

#### Первый вариант интерфейса программы

Первый вариант интерфейса (рис.2.8) изначально представлял, как будут расположены все игровые элементы, а также счетчики очков и жизней. Для управления игровой платформой используются стрелки на клавиатуре (влево и вправо).

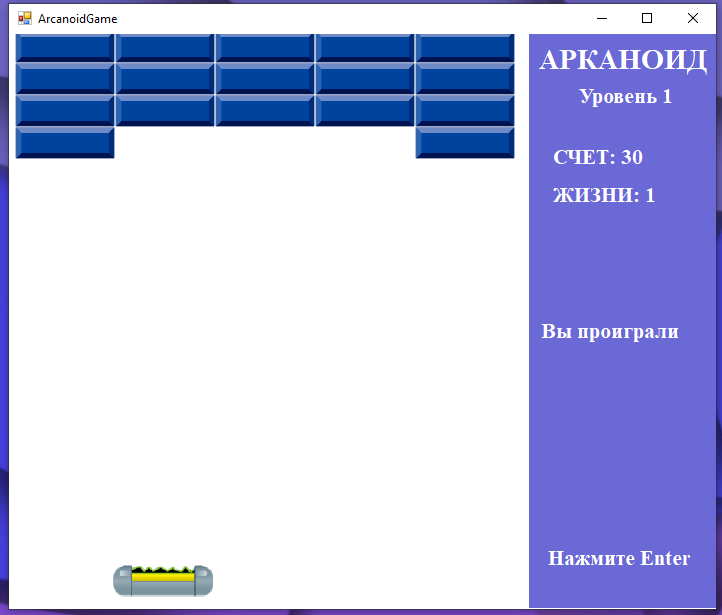


Рис.2.8. Первый вариант интерфейса программы

К достоинствам данного варианта можно отнести:

* простоту и яркую палитру цветов;
* счётчики жизней и очков расположены на видном месте, что не даст игроку отвлекаться во время игры;
* низкая нагрузка на ресурсы компьютера.

К недостаткам данного решения можно отнести:

* отсутствие главного меню и правил игры;
* неудачное расположение сообщений о проигрыше или выигрыше;
* в целом интерфейс за счёт своей простоты кажется скучным и, скорее всего, может вызвать дискомфорт у пользователя при долгом сеансе.

На основе проведенного анализа данного пользовательского интерфейса его достоинства были учтены, а недостатки исправлены в следующем варианте, ставшем финальным решением.

#### Второй вариант интерфейса программы

Как уже было сказано выше, данный интерфейс был разработан с учетом ошибок первого варианта. При его создании были заготовлены макеты, стилистика которых погружает игрока в ретро атмосферу. Сама игровая область выглядит интереснее и красивей за счёт нового шрифта, а игровые сообщения теперь появляются в центре экрана (рис.2.9). В программе появилось главное меню с космической тематикой (рис.2.10). В меню теперь находятся правила игры и конструктор уровней (рис.2.11). Но к недостаткам можно отнести большую нагрузку на ресурсы оперативной памяти.

Таким образом, благодаря ранее рассмотренным программным решениям и анализу первого варианта интерфейса был создан конечный интерфейс программы, который исправил все недостатки и сохранил в себе выявленные достоинства прошлых решений.

Рис.2.9. Второй вариант интерфейса программы

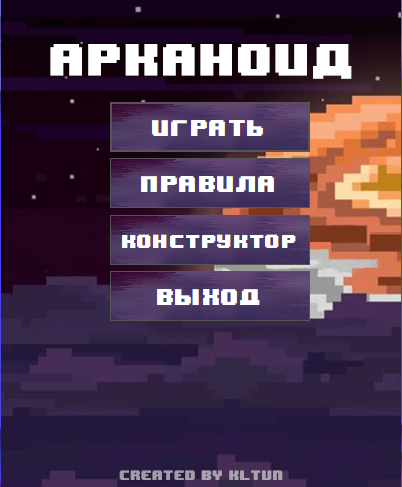
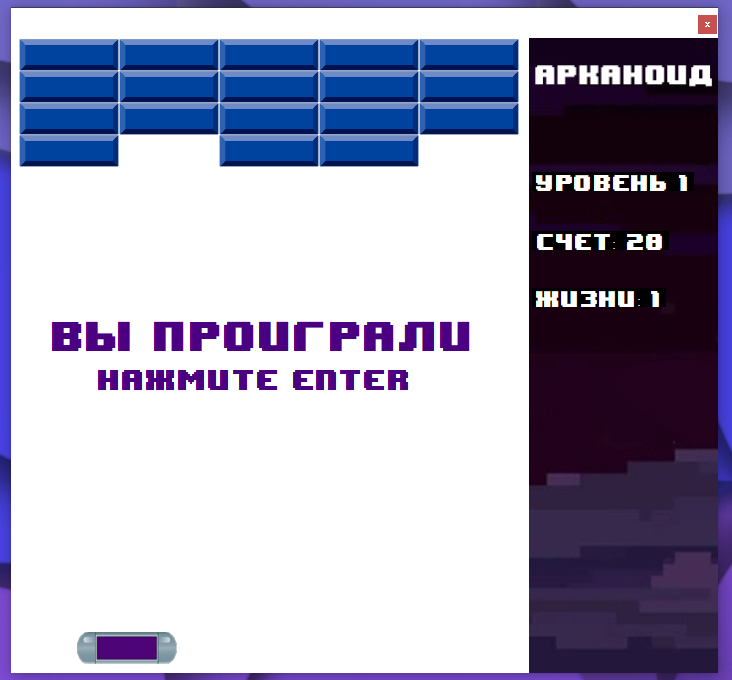


Рис.2.10. Главное меню игры

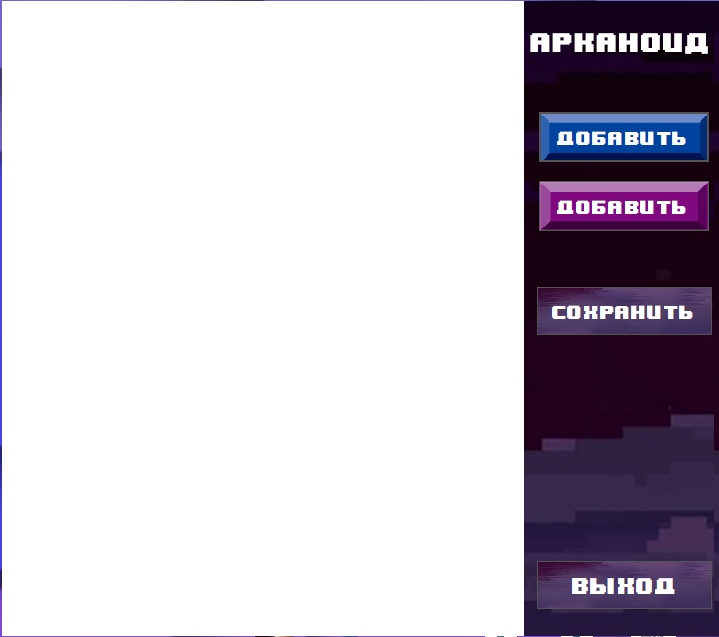


Рис.2.11. Конструктор пользовательских уровней

## ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

### Описание структуры и состав программной среды

В состав программной системы входят классы, содержащие конструкторы, свойства и методы.

Класс Game – отвечает за корректную работу всей игры.

Свойства:

private bool GameIsOver { get; set; } - окончание игры.

private bool NextLevel { get; set; } – прохождение начального уровня.

private int Score { get; set; } – счетчик очков.

private int Lives { get; set; } – счетчик жизней.

Методы:

private void GameSetup() – начальные настройки игры.

private void GenerateBlocks() – генерация блоков.

private void RemoveBlocks() – очистка игрового поля от блоков.

public void GameTime(object sender, EventArgs e) – отвечает за логику игры (проигрыш, выигрыш, продолжение игры) и за полёт игрового мяча.

public void Collision(object sender, EventArgs e) – отслеживает столкновение мяча о игровые элементы.

private void GenerateBonus() – отвечает за генерацию бонусов.

public void KeyIsDown(object sender, KeyEventArgs e) и public void KeyIsUp(object sender, KeyEventArgs e) – обработчики нажатия на стрелки влево и вправо для управления игровой платформой.

Класс Platform – представляет игровую платформу, которой управляет игрок.

Свойства:

public int PlatformSpeed { get; } – скорость платформы.

public int PlatformWidth { get; } – ширина платформы.

public bool MoveLeft { get; set; } – разрешает перемещение платформы влево.

public bool MoveRight { get; set; } – разрешает перемещение платформы вправо.

Конструктор:

public Platform()

{

PlatformSpeed = 12;

PlatformWidth = 100;

}

Класс Ball – представляет игровой мяч.

Свойства:

public int BallSpeedX { get; set; } – скорость мяча по оси X.

public int BallSpeedY { get; set; } – скорость мяча по оси Y.

Конструктор:

public Ball()

{

BallSpeedX = 5;

BallSpeedY = 5;

}

Абстрактный класс Bonus – представляет игровые бонусы.

Свойства:

public int BonusSpeed { get; } – скорость падение бонуса.

public int BonusHeight { get; } – высота изображения бонуса.

public int BonusWidth { get; } – ширина изображения бонуса.

public string BonusBackGround { get; } – изображение бонуса.

Конструктор:

public Bonus(string bonusBackGround)

{

BonusSpeed = 3;

BonusHeight = 25;

BonusWidth = 26;

BonusBackGround = bonusBackGround;

}

От этого класса наследуются классы, представляющие конкретные игровые бонусы.

Класс LifeBonus – класс, наследуемый от класса Bonus, представляющий бонус дополнительной жизни.

Свойства:

public int ExtraLife { get; } – дополнительная игровая жизнь.

Конструктор:

public LifeBonus() : base(@"Images\LifeBonus.png")

{

ExtraLife = 1;

}

Класс ScoreBonus – класс, наследуемый от класса Bonus, представляющий бонус дополнительных очков.

Свойства:

public int ExtraScore { get; } – дополнительные игровые очки.

Конструктор:

public ScoreBonus() : base(@"Images\ScoreBonus.png")

{

ExtraScore = 50;

}

Класс PlatformBonus – класс, наследуемый от класса Bonus, представляющий бонус увеличенной платформы.

Свойства:

public int ExtraWidth { get; } – дополнительная ширина платформы.

Конструктор:

public PlatformBonus() : base(@"Images\PlatBonus.png")

{

ExtraWidth = 50;

}

Класс GameArea – представляет игровую область, на которой размещаются блоки, платформа и мяч.

public int BlockHeight { get; } – высота игровой области.

public int BlockWidth { get; } – ширина игровой области.

public Blocks()

{

BlockHeight = 32;

BlockWidth = 100;

}

Класс Blocks – представляет игровые блоки.

public int BlockHeight { get; } – высота блока.

public int BlockWidth { get; } – ширина блока.

public Blocks()

{

BlockHeight = 32;

BlockWidth = 100;

}

### Разработка алгоритмов отдельных подзадач

Важной задачей является создание и загрузка файла, содержащего различные блоки и их расположение. Это необходимо для создания пользователем собственных игровых уровней. Этот алгоритм реализован в конструкторе уровней (рис.3.1).

Пользователь, нажимая на кнопки, добавляет на холст блоки. В созданный ранее документ записывается «0», если пользователь добавил синий блок, и «1», если пользователь добавил фиолетовый блок. По итогу в документе оказывается массив нулей и единиц, представляющих игровые блоки (рис.3.2).

После прохождение игроком начального уровня, происходит переход на следующий уровень, созданный пользователем (рис.3.3).

Рис.3.1. Конструктор игровых уровней

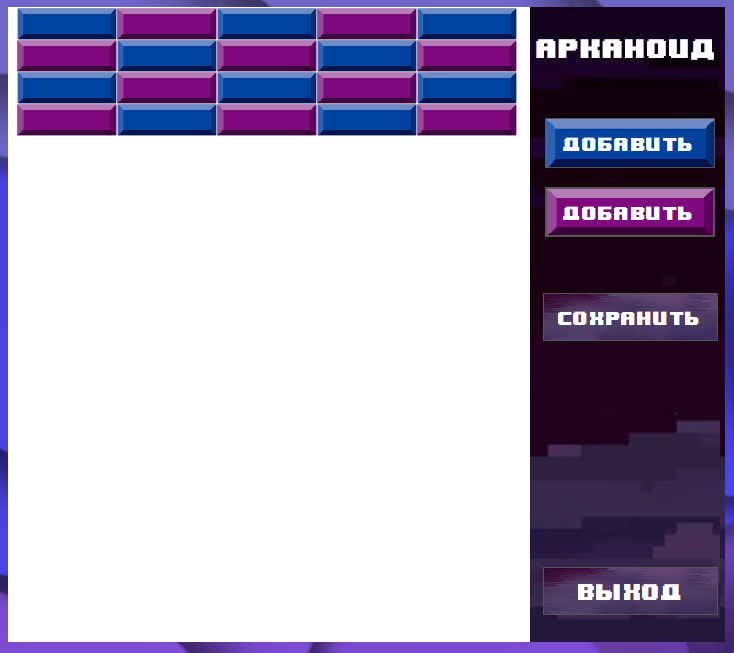
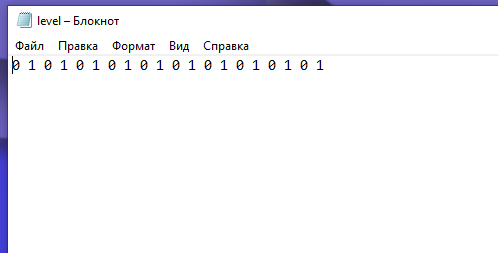


Рис.3.2. Документ, представляющий пользовательский уровень

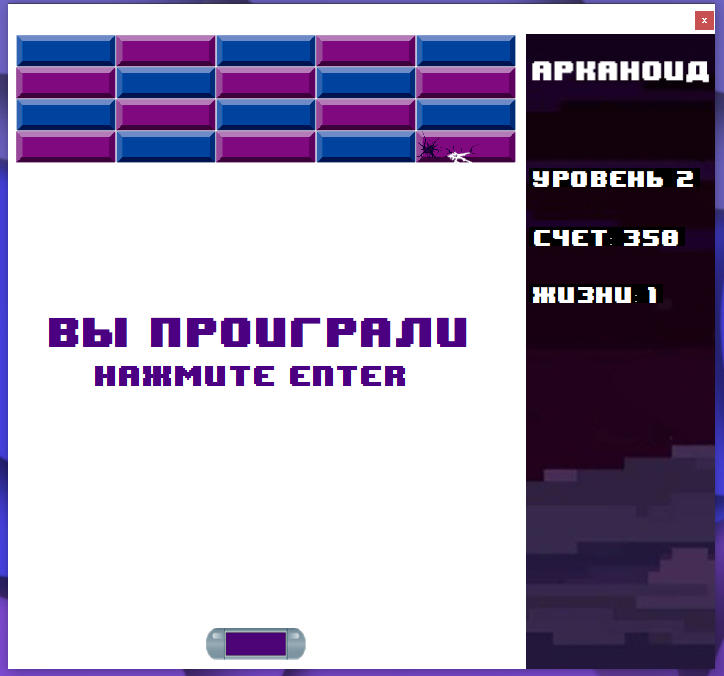


Рис.3.3.Отображение созданного уровня во время игры

### Руководство пользователя

Программа «игра Арканоид» является реализацией игры «Арканоид», созданной ещё в 1986 году компанией Taito. Суть игры проста: игроку нужно отбивать ракеткой мяч и сбивать им блоки, располагающиеся вверху игровой области, после разрушения которых, игрок зарабатывает очки, а также могут генерироваться случайные игровые бонусы. Управление ракеткой, или платформой, производится путем нажатия стрелок «влево» и «вправо» на клавиатуре. Цель игры – разрушить все блоки, не давая при этом мячу упасть вниз, иначе игрок потерпит поражение.

Эти правила описаны в самой программе, пользователь может прочесть их, нажав на кнопку «Правила» в главном меню игры (рис.3.4).

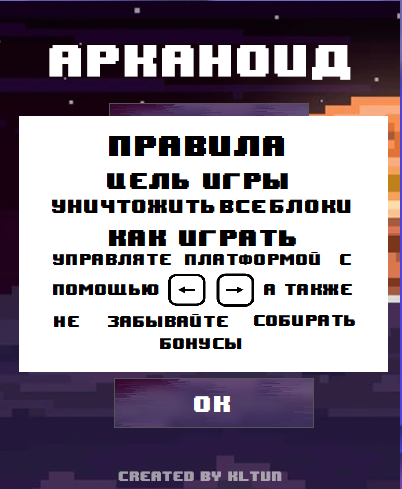


Рис.3.4. Правила игры «Арканоид»

### Подготовка программы к работе и условия её эксплуатации

Проект программы состоит из exe-файла и компонентов, необходимых для её работы. Его установка осуществляется с помощью копирования папки проекта и её компонентов на жёсткий диск пользователя, при этом должны быть соблюдены следующие системные требования для запуска:

* процессор Intel Core i3 или его аналоги;
* не менее 4 Гб оперативной памяти;
* объем видеопамяти от 2 Гб и выше;
* 64 – разрядная операционная система Windows 10 версии.
* устройства, необходимые для взаимодействия пользователя с программой, такие как клавиатура и мышь.

Указанные системные требования являются рекомендуемые, что значит, если компьютер пользователя не удовлетворяет данным характеристикам, то разработчик не может гарантировать полностью корректной работы программного комплекса.

Для удобства использования программы можно создать ярлык exe-файла в папке с проектом и переместить его на рабочий стол для быстрого запуска (рис.3.5).

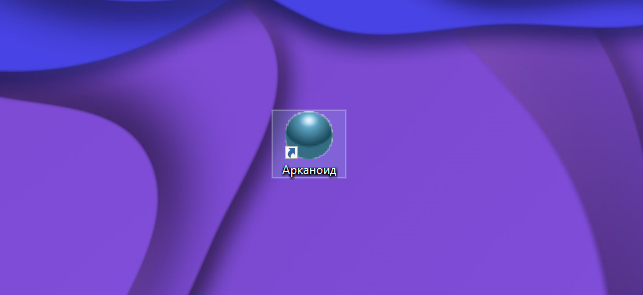


Рис.3.5. Ярлык программы на рабочем столе

## ЭКСПЕРИМЕНТАЛЬНЫЙ РАЗДЕЛ

### Виды контроля качества разрабатываемого ПО

Тестирование – очень важный и трудоёмкий этап процесса разработки программного обеспечения, так как правильно тестирование позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ.

Для повышения качества тестирования рекомендуется соблюдать следующие основные принципы:

* предполагаемые результаты тестирования должны быть известны ещё до начала тестов;
* следует избегать тестирования программы самим автором;
* необходимо досконально изучать результаты каждого проведенного теста;
* необходимо проверять действие программы на неверных данных;
* необходимо проверять программу на неожиданные побочные эффекты с использованием неверных данных.

Следует также иметь в виду, что вероятность наличия необнаруженных ошибок в части программы пропорциональны количеству ошибок, уже найденных в этой части.

Также различают два подхода к формированию тестов: структурный и функциональный. Каждый из этих подходов имеет свои особенности и области применения.

Структурный подход базируется на том, что известна структура тестируемого программного обеспечения, в том числе и его алгоритмы. В этом случае тесты строят так, чтобы проверить правильность реализации заданной логики в коде программы.

Функциональный подход основывается на том, что структура программного обеспечения неизвестна, в этом случае тесты строят, опираясь на функциональные спецификации. Этот подход называют также подходом, управляемым данными, так как при его использовании тесты строят на базе различных способов декомпозиции множества данных.

Наборы тестов, полученные в соответствии с методами этих подходов, обычно объединяют, обеспечивая всестороннее тестирование программного обеспечения.

### Методика проведения и результаты тестирования

Во время тестирования программы были использованы следующие принципы:

* необходимо подбирать тесты не только для правильных, но и для неверных данных;
* предполагаемые результаты тестирования должны быть известны ещё до начала тестов;
* каждый тест должен быть досконально изучен и перепроверен, чтобы не пропустить малозаметную на первый взгляд ошибку в программе;
* по возможности, следует избегать тестирования самим автором программы;
* тестирование не должно превращаться в рутинную работу, требуется проявлять повышенный интерес и подходить к этому процессу творчески.

В результате была выявлена следующая проблема: при завершении игроком начального, а затем и пользовательского уровня, игра не заканчивалась. Завершение игры происходит в момент, когда на игровой области нет ни одного блока. Оказалось, в документе, содержащий пользовательский игровой уровень, находилась пустая строка, из-за которой программа добавляет на игровую область пустой объект. Эта проблема была решена с помощью метода, который убирает все пустые строки из документа.

### Методы и способы устранения ошибок

Отладка программы – один из самых сложных этапов разработки программного обеспечения. Это процесс локализации и исправления ошибок, обнаруженных при тестировании. Локализацией называют процесс определение оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить её причину, то есть, определить оператор, содержащий ошибку. Эти причины могут быть как очевидными, так и глубоко скрытыми.

В соответствии с этапом обработки, на котором появляются ошибки, различают:

* синтаксические ошибки – ошибки, фиксируемые компилятором при выполнении синтаксического и семантического анализа программы;
* ошибки компоновки – ошибки, обнаруженные компоновщиком (редакторов связи) при объединении программных модулей;
* ошибки выполнения – ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

Существуют две взаимодополняющие технологии отладки.

Использование отладчиков – программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.

Вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода – на экран, принтер или в файл. Вывод отладочных сведений в файл называется журналированием.

Типичный цикл разработки, за время жизни программы многократно повторяющийся, выглядит примерно так:

* программирование — внесение в программу новой функциональности, исправление ошибок в имеющейся;
* тестирование (ручное или автоматизированное; программистом, тестером или пользователем) — обнаружение факта ошибки;
* воспроизведение ошибки — выяснение условий, при которых ошибка случается, это может оказаться непростой задачей при программировании параллельных процессов и при некоторых необычных ошибках;
* отладка — обнаружение причины ошибки.

Способности программиста к отладке — это, по-видимому, важнейший фактор в обнаружении источника проблемы, но сложность отладки сильно зависит от используемого языка программирования и инструментов, в частности, отладчиков.

### Отладка выявленных ошибок, обнаруженных при тестировании

При создании уровня в конструкторе была обработана ситуация с помощью условий if и else, когда пользователь пытается сохранить уровень, не добавив ни одного блока на холст.

if(DrawArea.Controls.Count == 0)

{

MessageBox.Show("Уровень не может быть пустым!");

}

else

{

FileStream fs = new FileStream("level.txt", FileMode.Open, FileAccess.ReadWrite);

fs.SetLength(0);

fs.Close();

StreamWriter writer = new StreamWriter("level.txt", true, Encoding.Default);

writer.Write(map.Trim());

writer.Close();

}

Также, при загрузке игрового уровня из документа, может возникнуть ошибка, если такого документа не существует или его название было изменено. Для обработки этой ошибка необходимо добавить исключительную ситуацию с помощью блоков try и catch.

string blocks;

try

{

StreamReader reader = new StreamReader("level.txt", Encoding.Default);

blocks = reader.ReadToEnd();

reader.Close();

string[] x = blocks.Split(null);

int row = 0;

int top = 0;

int left = 12;

for (int i = 0; i < x.Length; i++)

{

gameBlocks.blocks = new PictureBox

{

Height = gameBlocks.BlockHeight,

Width = gameBlocks.BlockWidth

};

if (row < 5)

{

if (x[i] == "0")

{

gameBlocks.blocks.Tag = "Blocks";

gameBlocks.blocks.BackgroundImage = Image.FromFile(@"Images\BlueBlock.png");

}

if (x[i] == "1")

{

gameBlocks.blocks.Tag = "HardBlocks";

gameBlocks.blocks.BackgroundImage = Image.FromFile(@"Images\PurpleBlock.png");

}

gameArea.AreaView.Controls.Add(gameBlocks.blocks);

gameBlocks.blocks.Left = left;

gameBlocks.blocks.Top = top;

row++;

left = left + 100;

}

if (row == 5)

{

top += 32;

left = 12;

row = 0;

}

}

}

catch(Exception ex)

{

MessageBox.Show($"Ошибка {ex.Message}! Файл не найден!");

}

## ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта был разработан программный комплекс «игра Арканоид». Его назначение состоит в том, чтобы пользователь мог приятно провести время с пользой для своего здоровья, снять накопленное напряжение и развить его внимание, моторику и быстроту реакции.

Перед его разработкой был проведен анализ предметной области и сравнительный обзор уже существующих программных решений, что позволило выявить набор функциональных требований к разрабатываемому программному продукту.

Разработаны функциональная схема работы программы, диаграмма потоков данных, диаграмма объектов и иерархия классов, для описания используемых методов и алгоритмов была разработана блок-схема программы. Также было разработано несколько вариантов пользовательского интерфейса, после анализа, которых был выявлен его конечный вариант.

Для пользователей, которые только знакомятся с игрой, было разработано руководство пользователя.

Программный продукт был протестирован на наличие ошибок, в процессе их обнаружения производилась их отладка.

Таким образом, в ходе выполнения курсового проекта был создан программный продукт, в котором были учтены все приведенные требования к графическим и функциональным возможностям, также, при необходимости, данная программа может дополняться новыми компонентами.

Был приобретен опыт в создании программного кода с использованием технологий объектно-ориентированного программирования, получены новые знания о различных инструментах языка программирования C#.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Буч, Г. Введение в UML от создателей языка / Г.Буч, Д.Рамбо, И.Якобсон. – ДМК Пресс, 2015. – 496 с.
2. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений. – Вильямс, 2017. – 720 с.
3. Зыков, С.В. Программирование. Объектно-ориентированный подход: учебник и практикум для академического бакалавриата. – М.: Издательство Юрайт, 2019. – 155 с.
4. Лаврищева, Е.М. Программная инженерия и технологии программирования сложных систем: учебник для вузов. – М.: Издательство Юрайт, 2019. – 432 с.
5. Либерти, Д. Программирование C#. – 2-е изд.: ЛитРес, 2017. – 679 с.
6. Павловская, Т.А. C#. Программирование на языке высокого уровня. – СПБ.: Питер, 2016. – 432 с.
7. Пышкин, Е.В. Основные концепции и механизмы объектно-ориентированного программирования. Учебник для вузов. – 2-е изд. – Изд.: БХВ-Петербург, 2005. – 640 с.
8. Троелсен, Э. Язык программирования C#и платформы .NET и .NET Core. – Диалектика, 2018. – 1328 с.
9. Арканоид. Статья на википедии [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Arkanoid
10. ГОСТ Р 7.0.5-2008. Библиографическая ссылка. Общие требования и правила составления. – М.: Стандартинформ, 2008. – 20 с.
11. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Изд-во стандартов, 1991. – 20 с.
12. ГОСТ 2.105-95. ЕСКД. Общие требования к текстовым документам. – М.: Изд-во стандартов, 2007. – 31 с.
13. ГОСТ 7.32-2001. Отчет о научно-исследовательской работе. Структура и правила оформления.- Стандартинформ, 2008.- 17 с.
14. Р 50.1.028-2001. «Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования».- М.:Изд-во стандартов, 2002. - 54 с.

## ПРИЛОЖЕНИЯ

### Приложение 1. Листинг программы

Листинг игры «Арканоид»

using System;

using System.Drawing;

using System.IO;

using System.Text;

using System.Windows.Forms;

namespace Курсовой\_проект\_ООП

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

DoubleBuffered = true;

ScoreCounter.Text = "СЧЕТ: 0";

LivesCounter.Text = "ЖИЗНИ: 3";

Msg.Text = "ЧТОБЫ НАЧАТЬ";

Continue.Text = "Нажмите пробел";

}

private void MainForm\_Load(object sender, EventArgs e)

{

Game game = new Game(GameBall, GamePlatform, GameArea, GameTimer, ScoreCounter, Msg, GameLevel, LivesCounter, Continue);

KeyUp += game.StartGame;

GameTimer.Tick += game.GameTime;

GameTimer.Tick += game.Collision;

KeyDown += game.KeyIsDown;

KeyUp += game.KeyIsUp;

KeyUp += game.GoNext;

}

}

class Game

{

Platform gamePlatform = new Platform();

Ball gameBall = new Ball();

GameArea gameArea = new GameArea();

Blocks gameBlocks = new Blocks();

Bonus gameBonus;

Timer gameTimer = new Timer();

Timer timeBonus = new Timer();

readonly Label scoreCounter = new Label();

readonly Label msg = new Label();

readonly Label gameLevel = new Label();

readonly Label livesCounter = new Label();

readonly Label continueGame = new Label();

private Random randomBallSpeed = new Random();

private bool GameIsOver { get; set; }

private bool NextLevel { get; set; }

private int Score { get; set; }

private int Lives { get; set; }

private bool BonusIsLive = true;

public Game(PictureBox ballView, PictureBox platformView, PictureBox areaView, Timer gameTime, Label sCount, Label mess, Label gLevel, Label lCount, Label cont)

{

gameBall.BallView = ballView;

gamePlatform.PlatformView = platformView;

gameArea.AreaView = areaView;

gameTimer = gameTime;

scoreCounter = sCount;

msg = mess;

gameLevel = gLevel;

livesCounter = lCount;

continueGame = cont;

}

private void GameSetup()

{

GameIsOver = false;

NextLevel = false;

Score = 0;

Lives = 3;

continueGame.Text = "";

msg.Text = "";

gameLevel.Text = "Уровень 1";

gameBall.BallView.Left = gameArea.AreaWidth / 2;

gameBall.BallView.Top = gameArea.AreaHeight / 2;

gamePlatform.PlatformView.Left = gameArea.AreaWidth / 2;

gameTimer.Start();

}

public void StartGame(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Space)

{

GenerateBlocks();

GameSetup();

}

}

private void GenerateBlocks()

{

int row = 0;

int top = 0;

int left = 12;

for (int i = 0; i < 20; i++)

{

gameBlocks.blocks = new PictureBox

{

Height = gameBlocks.BlockHeight,

Width = gameBlocks.BlockWidth,

Tag = "Blocks",

BackgroundImage = Image.FromFile(@"Images\BlueBlock.png")

};

if (row == 5)

{

top += 32;

left = 12;

row = 0;

}

if (row < 5)

{

row++;

gameBlocks.blocks.Left = left;

gameBlocks.blocks.Top = top;

gameArea.AreaView.Controls.Add(gameBlocks.blocks);

left = left + 100;

}

}

}

private void GenerateLevel()

{

string blocks;

try

{

StreamReader reader = new StreamReader("level.txt", Encoding.Default);

blocks = reader.ReadToEnd();

reader.Close();

string[] x = blocks.Split(null);

int row = 0;

int top = 0;

int left = 12;

for (int i = 0; i < x.Length; i++)

{

gameBlocks.blocks = new PictureBox

{

Height = gameBlocks.BlockHeight,

Width = gameBlocks.BlockWidth

};

if (row < 5)

{

if (x[i] == "0")

{

gameBlocks.blocks.Tag = "Blocks";

gameBlocks.blocks.BackgroundImage = Image.FromFile(@"Images\BlueBlock.png");

}

if (x[i] == "1")

{

gameBlocks.blocks.Tag = "HardBlocks";

gameBlocks.blocks.BackgroundImage = Image.FromFile(@"Images\PurpleBlock.png");

}

gameArea.AreaView.Controls.Add(gameBlocks.blocks);

gameBlocks.blocks.Left = left;

gameBlocks.blocks.Top = top;

row++;

left = left + 100;

}

if (row == 5)

{

top += 32;

left = 12;

row = 0;

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка {ex.Message}! Файл не найден!");

}

}

private void RemoveBlocks()

{

foreach (Control x in gameArea.AreaView.Controls)

{

gameArea.AreaView.Controls.Clear();

}

}

public void GameTime(object sender, EventArgs e)

{

scoreCounter.Text = "СЧЕТ: " + Score;

livesCounter.Text = "ЖИЗНИ: " + Lives;

if (gamePlatform.MoveLeft == true && gamePlatform.PlatformView.Left > 0)

{

gamePlatform.PlatformView.Left -= gamePlatform.PlatformSpeed;

}

if (gamePlatform.MoveRight == true && gamePlatform.PlatformView.Left < gameArea.AreaView.Width - gamePlatform.PlatformView.Width)

{

gamePlatform.PlatformView.Left += gamePlatform.PlatformSpeed;

}

gameBall.BallView.Left += gameBall.BallSpeedX;

gameBall.BallView.Top += gameBall.BallSpeedY;

if (gameBonus != null)

{

gameBonus.BonusView.Top += gameBonus.BonusSpeed;

}

if (gameArea.AreaView.Controls.Count == 0 && NextLevel == false)

{

GameContinue();

Score += 100;

}

if (gameBall.BallView.Top > gameArea.AreaView.Height)

{

Lives--;

if (Lives <= 0)

{

GameOver("Вы проиграли");

}

else

{

gameBall.BallView.Left = gameArea.AreaWidth / 2;

gameBall.BallView.Top = gameArea.AreaHeight / 2;

gamePlatform.PlatformView.Left = gameArea.AreaWidth / 2;

}

}

if (gameArea.AreaView.Controls.Count == 0 && NextLevel == true)

{

GameOver("Вы победили");

}

}

public void Collision(object sender, EventArgs e)

{

if (gameBall.BallView.Bounds.IntersectsWith(gamePlatform.PlatformView.Bounds))

{

gameBall.BallSpeedY = randomBallSpeed.Next(6, 10) \* -1;

if (gameBall.BallSpeedX < 0)

{

gameBall.BallSpeedX = randomBallSpeed.Next(6, 10) \* -1;

}

else

{

gameBall.BallSpeedX = randomBallSpeed.Next(6, 10);

}

}

if (gameBall.BallView.Left < 0 || gameBall.BallView.Left > gameArea.AreaWidth - gameBall.BallView.Width)

{

gameBall.BallSpeedX = -gameBall.BallSpeedX;

}

if (gameBall.BallView.Top < 0 || gameBall.BallView.Top > gameArea.AreaHeight)

{

gameBall.BallSpeedY = -gameBall.BallSpeedY;

}

foreach (Control x in gameArea.AreaView.Controls)

{

if (x is PictureBox && (string)x.Tag == "Blocks")

{

if (gameBall.BallView.Bounds.IntersectsWith(x.Bounds))

{

Score += 10;

gameBall.BallSpeedY = -gameBall.BallSpeedY;

gameArea.AreaView.Controls.Remove(x);

GenerateBonus();

}

}

if (x is PictureBox && (string)x.Tag == "HardBlocks")

{

if (gameBall.BallView.Bounds.IntersectsWith(x.Bounds))

{

gameBall.BallSpeedY = -gameBall.BallSpeedY;

x.BackgroundImage = Image.FromFile(@"Images\PurpleBlockDestroyed.png");

x.Tag = "damageBlock";

}

}

else if (x is PictureBox && (string)x.Tag == "damageBlock")

{

if (gameBall.BallView.Bounds.IntersectsWith(x.Bounds))

{

Score += 10;

gameBall.BallSpeedY = -gameBall.BallSpeedY;

gameArea.AreaView.Controls.Remove(x);

}

}

if (x is PictureBox && (string)x.Tag == "ScoreBonus")

{

if (x.Top > gameArea.AreaView.Height)

{

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

}

if (gamePlatform.PlatformView.Bounds.IntersectsWith(x.Bounds))

{

Score += ((ScoreBonus)gameBonus).ExtraScore;

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

}

}

if (x is PictureBox && (string)x.Tag == "LifeBonus")

{

if (x.Top > gameArea.AreaView.Height)

{

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

}

if (gamePlatform.PlatformView.Bounds.IntersectsWith(x.Bounds))

{

if (Lives < 3)

{

Lives += ((LifeBonus)gameBonus).ExtraLife;

}

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

}

}

if (x is PictureBox && (string)x.Tag == "PlatformBonus")

{

if (x.Top > gameArea.AreaView.Height)

{

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

}

if (gamePlatform.PlatformView.Bounds.IntersectsWith(x.Bounds))

{

gamePlatform.PlatformView.Width = gamePlatform.PlatformWidth + ((PlatformBonus)gameBonus).ExtraWidth;

gamePlatform.PlatformView.BackgroundImage = Image.FromFile(@"Images\BigPlatform.png");

gameArea.AreaView.Controls.Remove(x);

BonusIsLive = true;

timeBonus.Interval = 10000;

timeBonus.Tick += TimeBonus\_Tick;

timeBonus.Start();

}

}

}

}

private void TimeBonus\_Tick(object sender, EventArgs e)

{

gamePlatform.PlatformView.Width = gamePlatform.PlatformWidth;

gamePlatform.PlatformView.BackgroundImage = Image.FromFile(@"Images\gamePlatform.png");

timeBonus.Stop();

}

private void GenerateBonus()

{

Random randBonus = new Random();

int random = randBonus.Next(1, 10);

if (random == 1)

{

if (BonusIsLive == true)

{

gameBonus = new ScoreBonus();

gameBonus.BonusView = new PictureBox

{

BackgroundImage = Image.FromFile(gameBonus.BonusBackGround),

Width = gameBonus.BonusWidth,

Height = gameBonus.BonusHeight,

Top = gameArea.AreaHeight / 2,

Left = gameArea.AreaWidth / 2,

Tag = "ScoreBonus"

};

}

BonusIsLive = false;

}

if (random == 2)

{

if (BonusIsLive == true)

{

gameBonus = new LifeBonus();

gameBonus.BonusView = new PictureBox

{

BackgroundImage = Image.FromFile(gameBonus.BonusBackGround),

Width = gameBonus.BonusWidth,

Height = gameBonus.BonusHeight,

Top = gameArea.AreaHeight / 2,

Left = gameArea.AreaWidth / 2,

Tag = "LifeBonus"

};

}

BonusIsLive = false;

}

if (random == 3)

{

if (BonusIsLive == true)

{

gameBonus = new PlatformBonus();

gameBonus.BonusView = new PictureBox

{

BackgroundImage = Image.FromFile(gameBonus.BonusBackGround),

Width = gameBonus.BonusWidth,

Height = gameBonus.BonusHeight,

Top = gameArea.AreaHeight / 2,

Left = gameArea.AreaWidth / 2,

Tag = "PlatformBonus"

};

}

BonusIsLive = false;

}

if (gameBonus != null) gameArea.AreaView.Controls.Add(gameBonus.BonusView);

}

private void GameContinue()

{

NextLevel = true;

gameLevel.Text = "Уровень 2";

gameBall.BallView.Left = gameArea.AreaWidth / 2;

gameBall.BallView.Top = gameArea.AreaHeight / 2;

gamePlatform.PlatformView.Left = gameArea.AreaWidth / 2;

RemoveBlocks();

GenerateLevel();

}

private void GameOver(string message)

{

GameIsOver = true;

gameTimer.Stop();

scoreCounter.Text = "СЧЕТ: " + Score;

msg.Text = message;

continueGame.Text = "Нажмите Enter";

}

public void KeyIsDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left)

{

gamePlatform.MoveLeft = true;

}

if (e.KeyCode == Keys.Right)

{

gamePlatform.MoveRight = true;

}

}

public void KeyIsUp(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left)

{

gamePlatform.MoveLeft = false;

}

if (e.KeyCode == Keys.Right)

{

gamePlatform.MoveRight = false;

}

}

public void GoNext(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter && GameIsOver == true)

{

RemoveBlocks();

GameSetup();

GenerateBlocks();

}

}

}

class Platform

{

public int PlatformSpeed { get; }

public int PlatformWidth { get; }

public bool MoveLeft { get; set; }

public bool MoveRight { get; set; }

public PictureBox PlatformView = new PictureBox();

public Platform()

{

PlatformSpeed = 12;

PlatformWidth = 100;

}

}

class Ball

{

public int BallSpeedX { get; set; }

public int BallSpeedY { get; set; }

public PictureBox BallView = new PictureBox();

public Ball()

{

BallSpeedX = 5;

BallSpeedY = 5;

}

}

class GameArea

{

public PictureBox AreaView = new PictureBox();

public int AreaWidth { get; }

public int AreaHeight { get; }

public GameArea()

{

AreaWidth = 516;

AreaHeight = 682;

}

}

class Blocks

{

public PictureBox blocks = new PictureBox();

public int BlockHeight { get; }

public int BlockWidth { get; }

public Blocks()

{

BlockHeight = 32;

BlockWidth = 100;

}

}

abstract class Bonus

{

public PictureBox BonusView = new PictureBox();

public int BonusSpeed { get; }

public int BonusHeight { get; }

public int BonusWidth { get; }

public string BonusBackGround { get; }

public Bonus(string bonusBackGround)

{

BonusSpeed = 3;

BonusHeight = 25;

BonusWidth = 26;

BonusBackGround = bonusBackGround;

}

}

class LifeBonus : Bonus

{

public int ExtraLife { get; }

public LifeBonus() : base(@"Images\LifeBonus.png")

{

ExtraLife = 1;

}

}

class ScoreBonus : Bonus

{

public int ExtraScore { get; }

public ScoreBonus() : base(@"Images\ScoreBonus.png")

{

ExtraScore = 50;

}

}

class PlatformBonus : Bonus

{

public int ExtraWidth { get; }

public PlatformBonus() : base(@"Images\PlatBonus.png")

{

ExtraWidth = 50;

}

}

}

Листинг конструктора пользовательских уровней

using System;

using System.Drawing;

using System.IO;

using System.Text;

using System.Windows.Forms;

namespace Курсовой\_проект\_ООП

{

public partial class LevelConstructor : Form

{

DrawBlocks block = new DrawBlocks();

string map;

public int rows = 0;

public int columns = 0;

public int count = 0;

public LevelConstructor()

{

InitializeComponent();

}

public void CreateBlueBlock()

{

block.blocksView = new PictureBox

{

Height = 32,

Width = 100,

Tag = "Blocks",

BackgroundImage = Image.FromFile(@"D:\Учеба\2 курс\ООП\Курсовая работа\Images\BlueBlock.png")

};

if (columns < 4)

{

if (rows < 5)

{

block.blocksView.Left = block.DirX;

block.blocksView.Top = block.DirY;

DrawArea.Controls.Add(block.blocksView);

block.DirX += block.blocksView.Width;

rows++;

}

if (rows == 5)

{

block.DirY += block.blocksView.Height;

block.DirX = 9;

columns++;

rows = 0;

}

}

}

public void CreatePurpleBlock()

{

block.blocksView = new PictureBox

{

Height = 32,

Width = 100,

Tag = "HardBlocks",

BackgroundImage = Image.FromFile(@"D:\Учеба\2 курс\ООП\Курсовая работа\Images\PurpleBlock.png")

};

if (columns < 4)

{

if (rows < 5)

{

block.blocksView.Left = block.DirX;

block.blocksView.Top = block.DirY;

DrawArea.Controls.Add(block.blocksView);

block.DirX += block.blocksView.Width;

rows++;

}

if (rows == 5)

{

block.DirY += block.blocksView.Height;

block.DirX = 9;

columns++;

rows = 0;

}

}

}

private void AddBlue\_Click(object sender, EventArgs e)

{

if (count < 20)

{

CreateBlueBlock();

map += "0 ";

}

count++;

}

private void AddPurple\_Click(object sender, EventArgs e)

{

if (count < 20)

{

CreatePurpleBlock();

map += "1 ";

}

count++;

}

private void SaveButton\_Click(object sender, EventArgs e)

{

if(DrawArea.Controls.Count == 0)

{

MessageBox.Show("Уровень не может быть пустым!");

}

else

{

FileStream fs = new FileStream("level.txt", FileMode.Open, FileAccess.ReadWrite);

fs.SetLength(0);

fs.Close();

StreamWriter writer = new StreamWriter("level.txt", true, Encoding.Default);

writer.Write(map.Trim());

writer.Close();

}

}

private void ConstructorExit\_Click(object sender, EventArgs e)

{

this.Close();

}

}

public class DrawBlocks

{

public int DirX { get; set; } // Координата X первого блока

public int DirY { get; set; } // Координата Y первого блока

public PictureBox blocksView = new PictureBox();

public PictureBox[] blocks = new PictureBox[20];

public DrawBlocks()

{

DirX = 9;

DirY = 1;

}

}

}

Листинг главного меню

using System;

using System.Windows.Forms;

namespace Курсовой\_проект\_ООП

{

public partial class ArcanoidMenu : Form

{

public ArcanoidMenu()

{

InitializeComponent();

}

private void Play\_Click(object sender, EventArgs e)

{

MainForm gameForm = new MainForm();

gameForm.Show();

}

private void MenuExit\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void GameRules\_Click(object sender, EventArgs e)

{

Rules.Visible = true;

Ok.Visible = true;

}

private void Ok\_Click(object sender, EventArgs e)

{

Rules.Visible = false;

Ok.Visible = false;

}

private void Constructor\_Click(object sender, EventArgs e)

{

LevelConstructor constructor = new LevelConstructor();

constructor.Show();

}

}

}

### Приложение 2. Графический интерфейс программы

